```
// Computer Program Listing Appendix Under 37 CFR 1.52(e)
// PatentSourceCode.txt
// Copyright (c) 2004. Sybase, Inc. All Rights Reserved.
Observer based transducer nodes.
 Class Name
  BNBasicTransducer
 Code Snippet
           public void setInput(BNRuleSetResolvedInput input) throws
BNGateInputValueInvalidException,
BNUnsupportedFunctionReturnTypeException,
    BNClassNotFoundException,
    BNNoSuchMethodException,
    BNIllegalAccessException,
    BNInvocationTargetException,
    BNInvalidClassMethodSeperatorException,
    BNMissingMethodNameException,
    BNMissingClassNameException,
    BNRuleSetInputMixedWithRuleInputException,
    BNUnsupportedDataTypeException,
    BNUnsupportedEvaluationException,
    BNParseException,
    BNIllegalDataConversionException,
    BNNotNULLInputOnSetOutputException,
    BNMissingMethodNameException,
    BNInvocationTargetException,
    BNGateMissingInputLinkException,
    BNMethodOverloadErrorException,
    BNResolverException {
    boolean inputChanged = setProperInput(input);        if (inputChanged) {
      evaluate(input);
    }
            _bn.getEngine().removeRecordedTransducer(this);
  }
Bundled transducers.
 Class Name
  BNBundledOprTransducers
 Code Snippet
  /*
    * This method will go through _transducers list and call
addRelatedTransducer()
    * on each of the recorded transducer by passing in the input transducer, and at
    * last it will add the input transducer into the _transducers
    *
    * INTERNAL USE ONLY
    * The reason why this method is public is that the API is
    * defined in an interface
    */
  public void addTransducer(BNOprTransducer transducer) throws
          BNUnsupportedOprRelException,
      BNParseException {
```

```
 if (transducer.getRightOperand() instanceof BNRuleMLInd) {
 // one variable
  switch (transducer.getOpr()) {
    case SESimpleEvaluatorConstant.PREDICATE_EQUAL:
      if (_relatedEqualTransducers.size() == 0) {
        _relatedEqualTransducers.add(transducer);
        _hasTransducer[0] = true;
      } else {
        insertTransducer(transducer, _relatedEqualTransducers, 0, _relatedEqualTransducers.size() - 1);
      }
      break;
    case SESimpleEvaluatorConstant.PREDICATE_NOT_EQUAL:
      if (_relatedNotEqualTransducers.size() == 0) {
        _relatedNotEqualTransducers.add(transducer);
        _hasTransducer[1] = true;
      } else {
      insertTransducer(transducer, _relatedNotEqualTransducers, 0, _relatedNotEqualTransducers.size() - 1);
      }
      break;
     case SESimpleEvaluatorConstant.PREDICATE_GREATER_THAN:
    case SESimpleEvaluatorConstant.PREDICATE_GREATER_THAN_OR_EQUAL:
 if (_relatedGreaterThanOrEqualTransducers.size() == 0) {
   _relatedGreaterThanOrEqualTransducers.add(transducer);
   _hasTransducer[2] = true;
 } else {
   insertTransducer(transducer, _relatedGreaterThanOrEqualTransducers, 0,
_relatedGreaterThanOrEqualTransducers.size() - 1);
 }
 break;
    case SESimpleEvaluatorConstant.PREDICATE_LESS_THAN:
    case SESimpleEvaluatorConstant.PREDICATE_LESS_THAN_OR_EQUAL:
 if (_relatedLessThanOrEqualTransducers.size() == 0) {
   _relatedLessThanOrEqualTransducers.add(transducer);
   _hasTransducer[3] = true;
 } else {
   insertTransducer(transducer, _relatedLessThanOrEqualTransducers, 0,
_relatedLessThanOrEqualTransducers.size() - 1);
 }
 break;
    case SESimpleEvaluatorConstant.PREDICATE_BEFORE_CALENDAR:
 if (_relatedBeforeTransducers.size() == 0) {
   _relatedBeforeTransducers.add(transducer);
   _hasTransducer[4] = true;
 } else {
   insertTransducerCalendar(transducer, _relatedBeforeTransducers, 0, _relatedBeforeTransducers.size() - 1);
 }
 break;
    case SESimpleEvaluatorConstant.PREDICATE_AFTER_CALENDAR:
 if (_relatedAfterTransducers.size() == 0) {
   _relatedAfterTransducers.add(transducer);
```

```
      _hasTransducer[5] = true;
    } else {
      insertTransducerCalendar(transducer, _relatedAfterTransducers, 0, _relatedAfterTransducers.size() - 1);
    }
    break;
         case SESimpleEvaluatorConstant.PREDICATE_EQUAL_IGNORE_CASE_STRING:
    if (_relatedEqualIgnoreCaseTransducers.size() == 0) {
      _relatedEqualIgnoreCaseTransducers.add(transducer);
      _hasTransducer[6] = true;
    } else {
      insertTransducerIgnoreCase(transducer, _relatedEqualIgnoreCaseTransducers, 0,
_relatedEqualIgnoreCaseTransducers.size() - 1);
    }
    break;
         case SESimpleEvaluatorConstant.PREDICATE_NOT_EQUAL_IGNORE_CASE_STRING:
           if (_relatedNotEqualIgnoreCaseTransducers.size() == 0) {
      relatedNotEqualIgnoreCaseTransducers.add(transducer);
      _hasTransducer[7] = true;
    } else {
      insertTransducerIgnoreCase(transducer, _relatedNotEqualIgnoreCaseTransducers, 0,
_relatedNotEqualIgnoreCaseTransducers.size() - 1);
    }
      break;
    case SESimpleEvaluatorConstant.PREDICATE_GREATER_THAN_IGNORE_CASE_STRING:
    case SESimpleEvaluatorConstant.PREDICATE_GREATER_THAN_OR_EQUAL_IGNORE_CASE_STRING:
      if (_relatedGreaterThanOrEqualIgnoreCaseTransducers.size() == 0) {
       _relatedGreaterThanOrEqualIgnoreCaseTransducers.add(transducer);
        _hasTransducer[8] = true;
      } else {
        insertTransducerIgnoreCase(transducer, _relatedGreaterThanOrEqualIgnoreCaseTransducers, 0,
_relatedGreaterThanOrEqualIgnoreCaseTransducers.size() - 1);
      }
      break;
    case SESimpleEvaluatorConstant.PREDICATE_LESS_THAN_IGNORE_CASE_STRING:
    case SESimpleEvaluatorConstant.PREDICATE_LESS_THAN_OR_EQUAL_IGNORE_CASE_STRING:
      if (_relatedLessThanOrEqualIgnoreCaseTransducers.size() == 0) {
        _relatedLessThanOrEqualIgnoreCaseTransducers.add(transducer);
        _hasTransducer[9] = true;
      } else {
        insertTransducerIgnoreCase(transducer, _relatedLessThanOrEqualIgnoreCaseTransducers, 0,
_relatedLessThanOrEqualIgnoreCaseTransducers.size() - 1);
      }
      break;
    default:
      _hasTransducer[10] = true;
      _otherTransducers.add(transducer);
         }
       } else {
       // two variables
       _hasTransducer[10] = true;
    _otherTransducers.add(transducer);
   }
       }
  /*
   * This method will go through _transducers list and call setInput() on each of
   * the recorded transducer by passing in the input if the input is not set yet
   * (the input and output could set through setOutput() in BNOprTransducer)
   */
         public void setInput(BNVarStringDirectInput input) throws
        BNGateInputValueInvalidException,
            BNUnsupportedFunctionReturnTypeException,
            BNClassNotFoundException,
            BNNoSuchMethodException,
           BNIllegalAccessException,
           BNInvocationTargetException,
          BNInvalidClassMethodSeperatorException,
            BNMissingMethodNameException,
           BNMissingClassNameException,
   BNRuleSetInputMixedWithRuleInputException,
    BNUnsupportedDataTypeException,
    BNUnsupportedEvaluationException,
    BNParseException,
    BNIllegalDataConversionException,
    BNGateMissingInputLinkException,
    BNNotNULLInputOnSetOutputException,
    BNMethodOverloadErrorException,
    BNResolverException {
   for (int i = 0; i < 11; i ++) {
    if (_hasTransducer[i] == true) {
     switch (i+1) {
      case 1:
   {
    String inputStr = input.getValue();
    BNOprTransducer newTRUETransducer = search(inputStr, _relatedEqualTransducersArray, 0,
_relatedEqualTransducersArray.length - 1);
    if (newTRUETransducer == null) {
     if (_equalTRUETransducer != null) {
      _equalTRUETransducer.setInput(input);
     }
    } else {
     if (newTRUETransducer != _equalTRUETransducer) {
      newTRUETransducer.setInput(input);
      if (_equalTRUETransducer != null) {
       _equalTRUETransducer.setOutput(false, newTRUETransducer.getInput());
    }
      _equalTRUETransducer = newTRUETransducer;
     }
    }
    break;
   }
   case 2:
   {
     boolean setRest = false;
     for (int j = 0; j < _relatedNotEqualTransducersArray.length; j ++) {
       BNOprTransducer transducer = (BNOprTransducer)(_relatedNotEqualTransducersArray[j]);
       if (setRest == false) {
         transducer.setInput(input);
         if (((BNBasicNode)(transducer)).isOutputCurrValueValid() == true &&
((BNBasicNode)(transducer)).getOutputPrevValue() == true && ((BNBasicNode)(transducer)).getOutputCurrValue() ==
false) {
       // this transducer has output change from true to false
    setRest = true;
         }
       } else {
         transducer.setOutput(true, transducer.getInput());
       }
     }
     break;
    }
   case 3:
   {
     boolean setRest = false;
     for (int j = 0; j < _relatedGreaterThanOrEqualTransducersArray.length; j ++) {
       BNOprTransducer transducer = (BNOprTransducer)(_relatedGreaterThanOrEqualTransducersArray[j]);
       if (setRest == false) {
         transducer.setInput(input);
         if (((BNBasicNode)(transducer)).isOutputCurrValueValid() == true &&
((BNBasicNode)(transducer)).getOutputCurrValue() == true) {
    if (((BNBasicNode)(transducer)).getOutputPrevValue() == false) {
           // this transducer has output change from false to true
     setRest = true;
    }
         }
       } else {
         transducer.setOutput(true, transducer.getInput());
       }
     }
     break;
   }
   case 4:
   {
     boolean setRest = false;
     for (int j = _relatedLessThanOrEqualTransducersArray.length - 1; j >= 0; j --) {
       BNOprTransducer transducer = (BNOprTransducer)(_relatedLessThanOrEqualTransducersArray[j]);
       if (setRest == false) {
         transducer.setInput(input);
         if (((BNBasicNode)(transducer)).isOutputCurrValueValid() == true &&
((BNBasicNode)(transducer)).getOutputCurrValue() == true) {
    if (((BNBasicNode)(transducer)).getOutputPrevValue() == false) {
     // this transducer has output change from false to true
```

```
      setRest = true;
    }
        }
      } else {
        transducer.setOutput(true, transducer.getInput());
      }
    }
    break;
   }
   case 5:
   {
    boolean setRest = false;
    for (int j = _relatedBeforeTransducersArray.length - 1; j >= 0; j --) {
      BNOprTransducer transducer = (BNOprTransducer)(_relatedBeforeTransducersArray[j]);
      if (setRest == false) {
        transducer.setInput(input);
        if (((BNBasicNode)(transducer)).isOutputCurrValueValid() == true &&
((BNBasicNode)(transducer)).getOutputCurrValue() == true) {
          if (((BNBasicNode)(transducer)).getOutputPrevValue() == false) {
     // this transducer has output change from false to true
      setRest = true;
    }
        }
      } else {
        transducer.setOutput(true, transducer.getInput());
      }
    }
    break;
   }
    case 6:
   {
    boolean setRest = false;
    for (int j = 0; j < _relatedAfterTransducersArray.length; j ++) {
      BNOprTransducer transducer = (BNOprTransducer)(_relatedAfterTransducersArray[j]);
      if (setRest == false) {
        transducer.setInput(input);
        if (((BNBasicNode)(transducer)).isOutputCurrValueValid() == true &&
((BNBasicNode)(transducer)).getOutputCurrValue() == true) {
    if (((BNBasicNode)(transducer)).getOutputPrevValue() == false) {
          // this transducer has output change from false to true
      setRest = true;
    }
        }
      } else {
        transducer.setOutput(true, transducer.getInput());
      }
    }
    break;
   }
   case 7:
```

```
  {
    String inputStr = input.getValue();
    BNOprTransducer newTRUETransducer = searchIgnoreCase(inputStr, _relatedEqualIgnoreCaseTransducersArray,
0, _relatedEqualIgnoreCaseTransducersArray.length - 1);
    if (newTRUETransducer == null) {
      if (_equalTRUETransducer != null) {
        _equalTRUETransducer.setInput(input);
      }
    } else {
      if (newTRUETransducer != _equalTRUETransducer) {
        newTRUETransducer.setInput(input);
        if (_equalTRUETransducer != null) {
          _equalTRUETransducer.setOutput(false, newTRUETransducer.getInput());
        }
        _equalTRUETransducer = newTRUETransducer;
      }
    }
    break;
      }
      case 8:
  {
    boolean setRest = false;
    for (int j = 0; j < _relatedNotEqualIgnoreCaseTransducersArray.length; j ++) {
      BNOprTransducer transducer = (BNOprTransducer)(_relatedNotEqualIgnoreCaseTransducersArray[j]);
      if (setRest == false) {
        transducer.setInput(input);
        if (((BNBasicNode)(transducer)).isOutputCurrValueValid() == true &&
((BNBasicNode)(transducer)).getOutputPrevValue() == false && ((BNBasicNode)(transducer)).getOutputCurrValue()
== true) {
          // this transducer has output change from true to false
          setRest = true;
        }
      } else {
        transducer.setOutput(true, transducer.getInput());
      }
    }
    break;
  }
  case 9:
  {
    boolean setRest = false;
    for (int j = 0; j < _relatedGreaterThanOrEqualIgnoreCaseTransducersArray.length; j ++) {
      BNOprTransducer transducer = (BNOprTransducer)(_relatedGreaterThanOrEqualIgnoreCaseTransducersArray[j]);
      if (setRest == false) {
        transducer.setInput(input);
        if (((BNBasicNode)(transducer)).isOutputCurrValueValid() == true &&
((BNBasicNode)(transducer)).getOutputCurrValue() == true) {
          if (((BNBasicNode)(transducer)).getOutputPrevValue() == false) {
            // this transducer has output change from false to true
     setRest = true;
  }
      }
     } else {
       transducer.setOutput(true, transducer.getInput());
     }
   }
   break;
  }
  case 10:
  {
   boolean setRest = false;
   for (int j = _relatedLessThanOrEqualIgnoreCaseTransducersArray.length - 1; j >= 0; j --) {
     BNOprTransducer transducer = (BNOprTransducer)(_relatedLessThanOrEqualIgnoreCaseTransducersArray[j]);
     if (setRest == false) {
       transducer.setInput(input);
       if (((BNBasicNode)(transducer)).isOutputCurrValueValid() == true &&
((BNBasicNode)(transducer)).getOutputCurrValue() == true) {
         if (((BNBasicNode)(transducer)).getOutputPrevValue() == false &&
((BNBasicNode)(transducer)).getOutputCurrValue() == true) {
      // this transducer has output change from false to true
    setRest = true;
  }
      }
     } else {
       transducer.setOutput(true, transducer.getInput());
     }
   }
   break;
  }
  case 11:
   for (int j = 0; j < _otherTransducersArray.length; j ++) {
     BNOprTransducer transducer = (BNOprTransducer)(_otherTransducersArray[j]);
       transducer.setInput(input);
   }
   break;
  }
    }
  }
       }
    .
 ////
 // protected member variables
 ////
 protected List _relatedEqualTransducers = new LinkedList();
 protected Object[] _relatedEqualTransducersArray = null;
 protected BNOprTransducer _equalTRUETransducer = null;
 protected List _relatedNotEqualTransducers = new LinkedList();
 protected Object[] _relatedNotEqualTransducersArray = null;
 protected List _relatedGreaterThanOrEqualTransducers = new LinkedList();
 protected Object[] _relatedGreaterThanOrEqualTransducersArray = null;
 protected List _relatedLessThanOrEqualTransducers = new LinkedList();
 protected Object[] _relatedLessThanOrEqualTransducersArray = null;
 protected List _relatedBeforeTransducers = new LinkedList();
 protected Object[] _relatedBeforeTransducersArray = null;
 protected List _relatedAfterTransducers = new LinkedList();
 protected Object[] _relatedAfterTransducersArray = null;
 protected List _relatedEqualIgnoreCaseTransducers = new LinkedList();
 protected Object[] _relatedEqualIgnoreCaseTransducersArray = null;
 protected BNOprTransducer _equalIgnoreCaseTRUETransducer = null;
 protected List _relatedNotEqualIgnoreCaseTransducers = new LinkedList();
 protected Object[] _relatedNotEqualIgnoreCaseTransducersArray = null;
 protected List _relatedGreaterThanOrEqualIgnoreCaseTransducers = new LinkedList();
 protected Object[] _relatedGreaterThanOrEqualIgnoreCaseTransducersArray = null;
 protected List _relatedLessThanOrEqualIgnoreCaseTransducers = new LinkedList();
 protected Object[] _relatedLessThanOrEqualIgnoreCaseTransducersArray = null;
 protected List _otherTransducers = new LinkedList();
 protected Object[] _otherTransducersArray = null;
 protected boolean[] _hasTransducer = new boolean[11];
 protected Map _tmpStringToDate = new HashMap();
 protected BNStructure _bn = null;
```

Weighted links.

Class Name

BNSmartTwoInputGate

Code Snippet

```
 public void init() throws BNNodeWithoutOutputLinkException,
     BNNodeWithoutInputLinkException {
   if (_tmpInputLinks != null) {
   // not init yet
     // call parent class' init()
     super.init();
     // get first and second link
     BNSmartNode link1 = (BNSmartNode)_tmpInputLinks.get(0);
     BNSmartNode link2 = (BNSmartNode)_tmpInputLinks.get(1);
     if (link1.totalOutputLinks() >= link2.totalOutputLinks()) {
     // first one has more weight
       // assign hight weight link
       _highWeightedInputLink = link1;
       // assign low weight link
       _lowWeightedInputLink = link2;
     } else {
     // first one has less weight
       // assign high weight link
       _highWeightedInputLink = link2;
       // assign low weight link
        _lowWeightedInputLink = link1;
     }
     // this member variable is done
     _tmpInputLinks = null;
   }
 }
 .
 ////
 // protected member variables
 ////
 protected BNSmartNode _highWeightedInputLink = null;
 protected BNSmartNode _lowWeightedInputLink = null;
 protected boolean _isHighWeightedInputLinkActive = true;
 protected boolean _isLowWeightedInputLinkActive = true;
 protected boolean _isHighWeightedInputLinkContributedToCount = false;
 protected boolean _isLowWeightedInputLinkContributedToCount = false;
```

Passivation.

Class Name

BNSmartTwoInputGate

Code Snippet

```
 /*
   * 1) move the sender from active output links to passivated output links
  * 2) increase _passvatedSignalCount
   * 3) if _passivatedSignalCount == _totalOutputLinks, set _status to
   *    be passivated, and send passivated signal to the active input link
   */
 void passivatingSignal(BNSmartGate sender) {
   // remove sender from active output links
   _outputLinksActive.remove(sender);
   // add sender to passivated output links
   _outputLinksPassivated.add(sender);
   // increase passivated count
   _passivatingSignalCount ++;
   if (_passivatingSignalCount == _totalOutputLinks) {
   // all output links are passivated
     // set status
     _status = STATUS_PASSIVATED;
     // set all inputs to be passivated
     if (_isHighWeightedInputLinkActive == true) {
       _isHighWeightedInputLinkActive = false;
       _highWeightedInputLink.passivatingSignal(this);
     }
     if (_isLowWeightedInputLinkActive == true) {
       _isLowWeightedInputLinkActive = false;
       _lowWeightedInputLink.passivatingSignal(this);
     }
       }
 }
 /*
   * 1) move the sender from passivated output links to active output links
   * 2) if current status is passivated, decrease _passivatedSignalCount,
   *    and send activating signals to both input links
   * 3) if current status is active, just decrease _passivatedSignalCount
   */
void activatingSignal(BNSmartGate sender) throws
         BNUnsupportedFunctionReturnTypeException,
        BNClassNotFoundException,
         BNNoSuchMethodException,
         BNIllegalAccessException,
          BNInvocationTargetException,
         BNInvalidClassMethodSeperatorException,
  BNMissingMethodNameException,
  BNMissingClassNameException,
  BNRuleSetInputMixedWithRuleInputException,
  BNUnsupportedDataTypeException,
  BNUnsupportedEvaluationException,
  BNParseException,
  BNIllegalDataConversionException,
  BNGateInputValueInvalidException,
  BNGateMissingInputLinkException,
  BNNotNULLInputOnSetOutputException,
  BNMethodOverloadErrorException,
  BNResolverException {
  // remove sender from passivated output links
  _outputLinksPassivated.remove(sender);
  // add sender to active output links
  _outputLinksActive.add(sender);
  // decrease passivated count
  _passivatingSignalCount --;
  if (_status == STATUS_PASSIVATED) {
    // change status to be active
    _status = STATUS_ACTIVE;
    // initial output value
    _outputCurrValue = false;
    _isOutputCurrValueValid = true;
    // set all inputs to be active
    _isLowWeightedInputLinkActive = true;
    _lowWeightedInputLink.activatingSignal(this);
    if (calculateResult(_lowWeightedInputLink, _isLowWeightedInputLinkContributedToCount, true) == true) {
      _isHighWeightedInputLinkActive = true;
      _highWeightedInputLink.activatingSignal(this);
      calculateResult(_highWeightedInputLink, _isHighWeightedInputLinkContributedToCount, false);
    }
  }
}
```

Class Name

BNSmartTwoInputGate

Code Snippet

```
/*
  * increase _passivatedSignalCount, if it equals to _totalOutputLinks,
  * set this transducer to passivated. And move the output link from
  * active list to passivated list
  */
void passivatingSignal(BNSmartGate sender) {
  // move output link from active links to passivated links
  if (sender instanceof BNANDGate) {
    _outputLinksANDGatesActive.remove(sender);
    _outputLinksANDGatesPassivated.add(sender);
  } else {
    _outputLinksORGatesActive.remove(sender);
    _outputLinksORGatesPassivated.add(sender);
  }
  // increase passivated count
  _passivatingSignalCount ++;
  if (_passivatingSignalCount == _totalOutputLinks) {
  // all output links are passivated and no rule is depending on
  // this transducer
    _status = STATUS_PASSIVATED;
  }
}
/*
  * decrease _passivatedSignalCount, and move the output link from passivated
  * list to active list. If this signal  turns the transducer from
  * passivated to active, do evaluation on the input value which is not
  * evaluated against yet
  */
void activatingSignal(BNSmartGate sender) throws
        BNUnsupportedFunctionReturnTypeException,
   BNClassNotFoundException,
   BNNoSuchMethodException,
   BNIllegalAccessException,
   BNInvocationTargetException,
   BNInvalidClassMethodSeperatorException,
   BNMissingMethodNameException,
   BNMissingClassNameException,
   BNRuleSetInputMixedWithRuleInputException,
   BNUnsupportedDataTypeException,
   BNUnsupportedEvaluationException,
   BNParseException,
   BNIllegalDataConversionException,
   BNGateMissingInputLinkException,
   BNNotNULLInputOnSetOutputException,
   BNGateInputValueInvalidException,
   BNMethodOverloadErrorException,
   BNResolverException {
  // move output link from passivated links to active links
  if (sender instanceof BNANDGate) {
    _outputLinksANDGatesPassivated.remove(sender);
    _outputLinksANDGatesActive.add(sender);
  } else {
    _outputLinksORGatesPassivated.remove(sender);
    _outputLinksORGatesActive.add(sender);
  }
  // decrease passivated count
  _passivatingSignalCount --;
  if (_status == STATUS_PASSIVATED) {
    // this signal turn this transducer from passivated
    // to active
      _status = STATUS_ACTIVE;
      if (isInputReadyForEvaluation() == true
     && _isOutputCurrValueValid == false) {
      // input was set but evaluation was never taken place
      // evalute on the input
        _outputCurrValue = evaluateExpression();
        _isOutputCurrValueValid = true;
      }
    }
  }
OR Nodes Support.
 Class Name
  BNSmartTwoInputORGate
 Code Snippet
  void setInput(BNSmartNode setter) throws
          BNUnsupportedFunctionReturnTypeException,
    BNClassNotFoundException,
    BNNoSuchMethodException,
    BNIllegalAccessException,
    BNInvocationTargetException,
    BNInvalidClassMethodSeperatorException,
    BNMissingMethodNameException,
    BNMissingClassNameException,
    BNRuleSetInputMixedWithRuleInputException,
    BNUnsupportedDataTypeException,
    BNUnsupportedEvaluationException,
    BNParseException,
    BNIllegalDataConversionException,
    BNGateMissingInputLinkException,
    BNGateInputValueInvalidException,
    BNNotNULLInputOnSetOutputException,
   BNMethodOverloadErrorException,
    BNResolverException {
           if (setter.isOutputCurrValueValid() == true) {
    // the output of setter is valid
      boolean isLowWeightedInputLink = (setter == _lowWeightedInputLink);
      // record prev output value
      _outputPrevValue = _outputCurrValue;
      _isOutputPrevValueValid = _isOutputCurrValueValid;
      // get input
      boolean input = setter.getOutputCurrValue();
              if (input == false) {
      // input is FALSE
        // increase count
        if (isLowWeightedInputLink == true && _isLowWeightedInputLinkContributedToCount == false) {
          _count ++;
          _isLowWeightedInputLinkContributedToCount = true;
        } else if (isLowWeightedInputLink == false && _isHighWeightedInputLinkContributedToCount == false) {
         _count ++;
         _isHighWeightedInputLinkContributedToCount = true;
            }
            if (_count == 2) {
        // all inputs are FALSE
         // activate the passivated link
         if (_isLowWeightedInputLinkActive == false || _isHighWeightedInputLinkActive == false) {
             // has passivated input link
          if (isLowWeightedInputLink == true) {
           if (_isHighWeightedInputLinkActive == false) {
            _isHighWeightedInputLinkActive = true;
                _highWeightedInputLink.activatingSignal(this);
                if ((_highWeightedInputLink.getOutputCurrValue() == false &&
_isHighWeightedInputLinkContributedToCount == false) || (_highWeightedInputLink.getOutputCurrValue() == true &&
_isHighWeightedInputLinkContributedToCount == true)) {
                 setInput(_highWeightedInputLink);
                } else if (_highWeightedInputLink.getOutputCurrValue() == false) {
                 // set curr output value
                 _outputCurrValue = false;
                 _isOutputCurrValueValid = true;
                 // send out signal
                 sendOutSignal();
                }
               }
              } else {
               if (_isLowWeightedInputLinkActive == false) {
                _isLowWeightedInputLinkActive = true;
                _lowWeightedInputLink.activatingSignal(this);
                if ((_lowWeightedInputLink.getOutputCurrValue() == false &&
_isLowWeightedInputLinkContributedToCount == false) || (_lowWeightedInputLink.getOutputCurrValue() == true &&
_isLowWeightedInputLinkContributedToCount == true)) {
                 setInput(_lowWeightedInputLink);
                } else if (_lowWeightedInputLink.getOutputCurrValue() == false) {
                 // set curr output value
                 _outputCurrValue = false;
                 _isOutputCurrValueValid = true;
            // send out signal
                 sendOutSignal();
                }
               }
          }
             } else {
                // no passivated input link
              // set curr output value
              _outputCurrValue = false;
              _isOutputCurrValueValid = true;
              // send out signal
              sendOutSignal();
             }
            } else {
              // not all inputs are FALSE
           // activate the passivated link
               if (isLowWeightedInputLink == true) {
                 if (_isHighWeightedInputLinkActive == false) {
             _isHighWeightedInputLinkActive = true;
                   _highWeightedInputLink.activatingSignal(this);
                   if ((_highWeightedInputLink.getOutputCurrValue() == false &&
_isHighWeightedInputLinkContributedToCount == false) || (_highWeightedInputLink.getOutputCurrValue() == true &&
_isHighWeightedInputLinkContributedToCount == true)) {
                     setInput(_highWeightedInputLink);
                   }
                 }
               } else {
                 if (_isLowWeightedInputLinkActive == false) {
                   _isLowWeightedInputLinkActive = true;
                   _lowWeightedInputLink.activatingSignal(this);
                   if ((_lowWeightedInputLink.getOutputCurrValue() == false &&
_isLowWeightedInputLinkContributedToCount == false) || (_lowWeightedInputLink.getOutputCurrValue() == true &&
_isLowWeightedInputLinkContributedToCount == true)) {
                     setInput(_lowWeightedInputLink);
                   }
                 }
               }
             }
           } else {
       // input is TRUE
         // desease the count
         _count --;
         if (isLowWeightedInputLink) {
           _isLowWeightedInputLinkContributedToCount = false;
         } else {
               _isHighWeightedInputLinkContributedToCount = false;
         }
         // set curr output value
         _outputCurrValue = true;
              _isOutputCurrValueValid = true;
         // passivate acitve input links
         if (isLowWeightedInputLink) {
           if (_isHighWeightedInputLinkActive == true) {
             _isHighWeightedInputLinkActive = false;
             _highWeightedInputLink.passivatingSignal(this);
           }
         } else {
           if (_isLowWeightedInputLinkActive == true) {
             _isLowWeightedInputLinkActive = false;
             _lowWeightedInputLink.passivatingSignal(this);
           }
         }
         if (_outputPrevValue == false) {
         // previous output value is true
```

```
        // propogate the change
        sendOutSignal();
      }
          }
         } else {
    // setter's output is not valid, throw exception
          throw new BNGateInputValueInvalidException();
    }
  }
```

Expression Factoring.

Class Name

BNRuleMLToBN

Code Snippet

```
 protected BNANDGate processAnd(Node andNode, String ruleHandle) throws RuleExecutionSetCreateException {
   BNANDGate retValue = null;
   // to record the BN nodes
   List bnNodes = new LinkedList();
   List existingTransducers = new LinkedList();
   List existingORGates = new LinkedList();
   List newTransducers = new LinkedList();
   List newORGates = new LinkedList();
   // get first child node of AND node
   Node childNode = andNode.getFirstChild();
   // go through each child node of AND node
   while (childNode != null) {
     if (childNode.getNodeName().compareTo(TAG_ATOM) == 0) {
     // it is ATOM node
       // get the transducer for this ATOM
       BNBasicNode bnNode = processAtom(childNode);
       // record this BN node
           if (bnNode.isNewCreated() == true) {
             newTransducers.add(bnNode);
       } else {
         insertNode(bnNode, existingTransducers);
       }
          } else if (childNode.getNodeName().compareTo(TAG_OR) == 0) {
          // it is OR node
            // get the OR gate for this OR
            BNORGate orGate = processOr(childNode, ruleHandle);
            if (orGate == null) {
            // error happened
              break;
            }
            // record this BN node
            if (((BNBasicNode)(orGate)).isNewCreated() == true) {
              newORGates.add(orGate);
            } else {
         insertNode((BNBasicNode)(orGate), existingORGates);
            }
          }
```

```
        childNode = childNode.getNextSibling();
      }
  bnNodes.addAll(existingTransducers);
  bnNodes.addAll(existingORGates);
  bnNodes.addAll(newTransducers);
  bnNodes.addAll(newORGates);
  List newANDGates = new LinkedList();
  if (bnNodes.size() != 0) {
      // hase BN nodes for the AND
        if (bnNodes.size() == 1) {
        // one input AND gate, AND gate is not needed
          // add related rule handle to this bn node direactly
          BNBasicNode bnNode = (BNBasicNode)(bnNodes.get(0));
          bnNode.addOutputRelatedRuleHandle(ruleHandle);
        } else {
          // more than one BN node
            // it is two input AND gate
            Object[] array = bnNodes.toArray();
            // record the two inputs
            List twoBNNodes = new LinkedList();
            // get first input
            twoBNNodes.add(array[0]);
            // go through rest of the input from position 1
            for (int i = 1; i < array.length; i ++) {
              // add second input node
              twoBNNodes.add(array[i]);
              try {
                // get the AND gate
                retValue = _bn.getANDGate(twoBNNodes, andNode);
                if (((BNBasicNode)(retValue)).isNewCreated() == true) {
                  newANDGates.add(retValue);
                }
              } catch (BNException e) {
                throw new RuleExecutionSetCreateException(e.getClass().getName(), e);
            }
            if (retValue == null) {
            // error happened
              break;
            }
            // renew the two input nodes track
            twoBNNodes = new LinkedList();
            // add the last AND gate
            twoBNNodes.add(retValue);
          }
          array = null;
        }
      }
  clearNewCreated(newTransducers);
  clearNewCreated(newORGates);
  newANDGates.remove(retValue);
  clearNewCreated(newANDGates);
  return retValue;
}
protected BNORGate processOr(Node orNode,
          String ruleHandle)
                      throws RuleExecutionSetCreateException{
  BNORGate retValue = null;
  // record the BN nodes
  List bnNodes = new LinkedList();
  List existingTransducers = new LinkedList();
  List existingANDGates = new LinkedList();
  List newTransducers = new LinkedList();
  List newANDGates = new LinkedList();
  // get the first node of the OR node
  Node childNode = orNode.getFirstChild();
  // go through each child node of OR node
  while (childNode != null) {
    if (childNode.getNodeName().compareTo(TAG_ATOM) == 0) {
    // it is ATOM node
      // get the transducer for this ATOM
      BNBasicNode bnNode = processAtom(childNode);
      // add the transducer to the BN node list
      if (bnNode.isNewCreated() == true) {
        newTransducers.add(bnNode);
      } else {
        insertNode(bnNode, existingTransducers);
      }
      } else if (childNode.getNodeName().compareTo(TAG_AND) == 0) {
    // it is AND node
      // get the AND gate
      BNANDGate andGate = processAnd(childNode, ruleHandle);
      if (andGate == null) {
      // error happened
 break;
      }
      // add the AND gate to the BN node list
      if (((BNBasicNode)(andGate)).isNewCreated() == true) {
 newANDGates.add(andGate);
      } else {
 insertNode((BNBasicNode)(andGate), existingANDGates);
      }
    }
    childNode = childNode.getNextSibling();
  }
  bnNodes.addAll(existingTransducers);
  bnNodes.addAll(existingANDGates);
  bnNodes.addAll(newTransducers);
  bnNodes.addAll(newANDGates);
  List newORGates = new LinkedList();
  if (bnNodes.size() != 0) {
```

```
  // has BN node
    if (bnNodes.size() == 1) {
    // one input OR gate, and gate is not needed
    // add related rule handle to this bn node direactly
      BNBasicNode bnNode = (BNBasicNode)(bnNodes.get(0));
      bnNode.addOutputRelatedRuleHandle(ruleHandle);
    } else {
    // more than one input OR gate
    // it is two input OR gate
    Object[] array = bnNodes.toArray();
    // record two inputs
    List twoBNNodes = new LinkedList();
    // add first input
    twoBNNodes.add(array[0]);
    // go through each BN node started at POSITION 1
    for (int i = 1; i < array.length; i ++) {
    // add second input
      twoBNNodes.add(array[i]);
      try {
        // get OR gate
 retValue = _bn.getORGate(twoBNNodes, orNode);
 if (((BNBasicNode)(retValue)).isNewCreated() == true) {
   newORGates.add(retValue);
 }
      } catch (BNException e) {
 throw new RuleExecutionSetCreateException(e.getClass().getName(), e);
      }
      if (retValue == null) {
      // error happened
        break;
      }
      // renew two input list
      twoBNNodes = new LinkedList();
      // add first input
      twoBNNodes.add(retValue);
    }
    array = null;
  }
}
clearNewCreated(newTransducers);
clearNewCreated(newANDGates);
newORGates.remove(retValue);
clearNewCreated(newORGates);
return retValue;
              }
```